7-17-1988

Diagnosis expert system with automated query to a process control system

Richard Winslow

Rochester Institute of Technology
School of Computer Science and Technology

Diagnosis Expert System with Automated Query to a Process Control System

By: Richard J. Winslow
July 17, 1988

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for
the degree of Master of Science in Computer Science

Approved by:

John A. Biles
John A. Biles

P. J. Meehan
Patrick J. Meehan

Peter G. Anderson
Peter G. Anderson

ABSTRACT

This thesis describes the development of a diagnosis expert system for an automated process control facility. To reduce the number of user responses to the expert system, a network interface was created between the expert system and the process control computer. This document focuses on the unique concerns associated with the development, validation, and implementation of an expert system that is directly interfaced to a process control system.

SUBJECT KEYWORDS

- Expert System
- Process Control
- Diagnosis System

PREFACE

COPYRIGHT CLAUSE

INTENDED AUDIENCE

This document is intended for readers who are knowledgeable in Computer Science and who have a basic understanding of Expert Systems.

DOCUMENT STRUCTURE

A summary of the chapters and appendices within this document are described below:

- INTRODUCTION - A statement about the need for specific expert system features when using them for process control applications.
- BACKGROUND - A discussion of existing process control expert systems, a description of the methods used to select the expert system shell, product descriptions for INSIGHT2+ and NEXPERT, and a justification for the expert system created during this thesis.
- IMPLEMENTATION - A description of the software design and the development approach; a list of major components in the knowledge base; and a project timeline.
- RESULTS - A description of the accomplishments.
- CONCLUSION - A discussion of the conclusions that can be drawn from the tool selection, the expert system design, the knowledge

acquisition process, and the testing phase.

- o SUMMARY
- o GLOSSARY - A brief description of important terms.

This document will reference many entities with trademarks. Below is a list of registered trademarks:

Insight2+ is a registered trademark of Level Five Research Inc., Indialantic, Florida.

VAX, VMS, PDP, DecNet are registered trademarks of Digital Equipment Corp., Maynard, MA.

IBM PC is a registered trademark of International Business Machines, Binghamton, NY.

NEXPERT is a registered trademark of Neuron Data, Palo Alto, CA.

TABLE

OF

CONTENTS

CHAPTER 1

INTRODUCTION

In recent years, industry has become aware of the potential of expert systems, which has produced a wave of expert system development. But for an expert system to be viable in industry, it must be able to access information outside the expert system.

An expert system should be able to communicate with the operating system, existing application software, programming languages, and local and distributed databases (See Figure 1.1). This is known as an integrated expert system. A medium or large standalone diagnosis expert system with all of its knowledge internal, requires an excessive amount of information to be entered by the end user, which can greatly reduce the effectiveness of the system. An integrated expert system, however, can divert a portion of its queries to information outside itself. Thus, it can have a smaller, more flexible rule base by acquiring values of objects and facts from a database; it can address applications that have dynamic data; and it can leave data administration to the DBMS and algorithmic calculations to conventional programming languages or existing applications. An

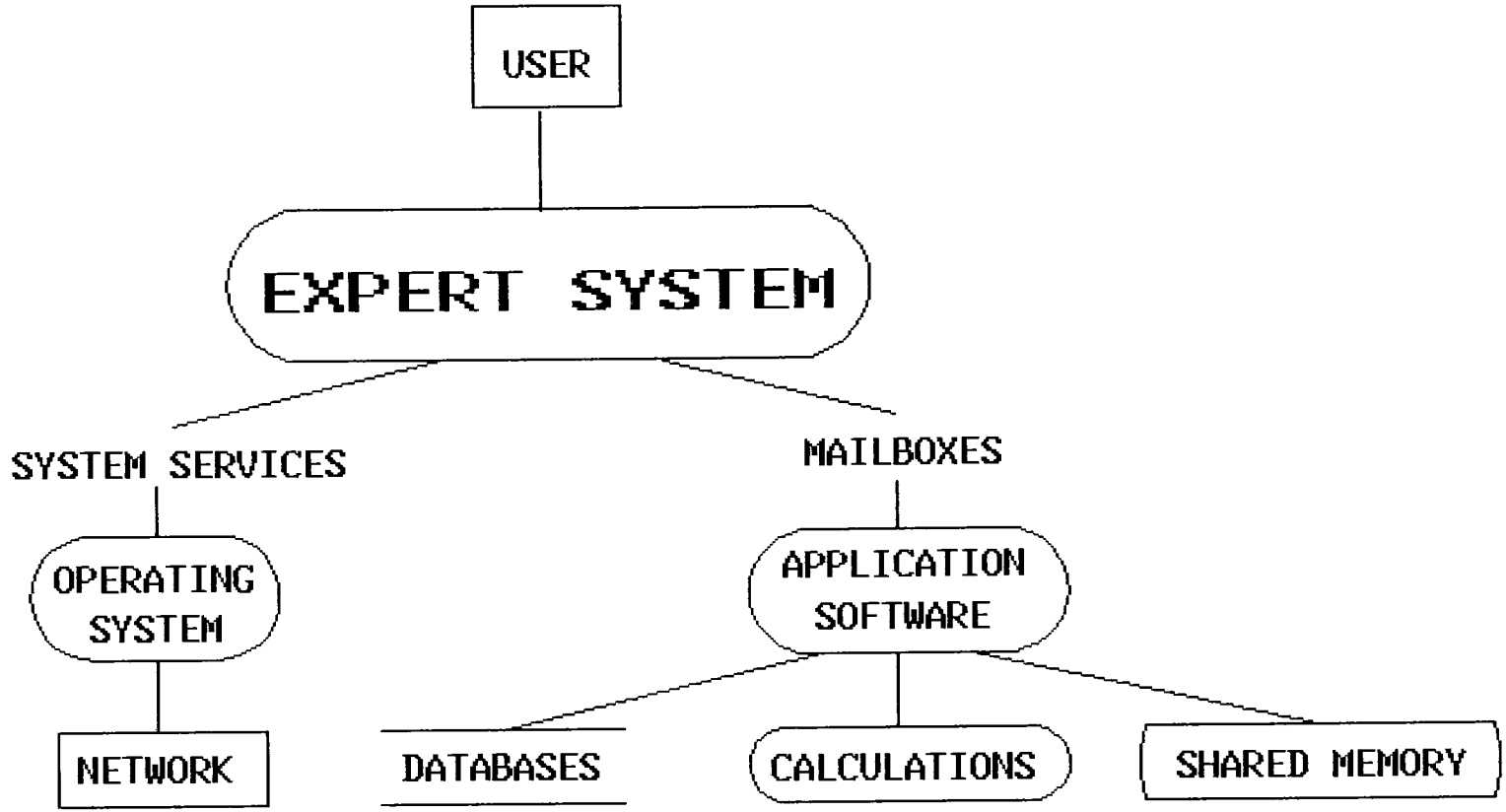

USER
EXPERT SYSTEM
SYSTEM SERVICES
MAILBOXES
OPERATING SYSTEM
APPLICATION SOFTWARE
NETWORK
DATABASES
CALCULATIONS
SHARED MEMORY

FIGURE 1.1

expert system that is fully integrated into its environment becomes an indispensable tool in the commercial marketplace.

The commercial market has also been asking for real-time expert systems for process monitoring and control (See Figure 1.2). Integrated expert systems make this possible. Real-time expert systems work with meta-knowledge above the level of a conventional process control system. The conventional control system uses well-defined PID control algorithms to maintain a single or cascaded control loop (e.g. temperature). The expert system can form an adaptive control scheme to optimize the control loop or orchestrate a multi-loop process control system (e.g. temperature, pH, flowrate).

The BMCS expert system developed for this thesis is fully integrated with the application through interfaces to the operating system, to the remote application computer, and the distributed databases. However the BMCS expert system merely warns the user if the process is poorly tuned, it does not currently function as a real-time adaptive controller.

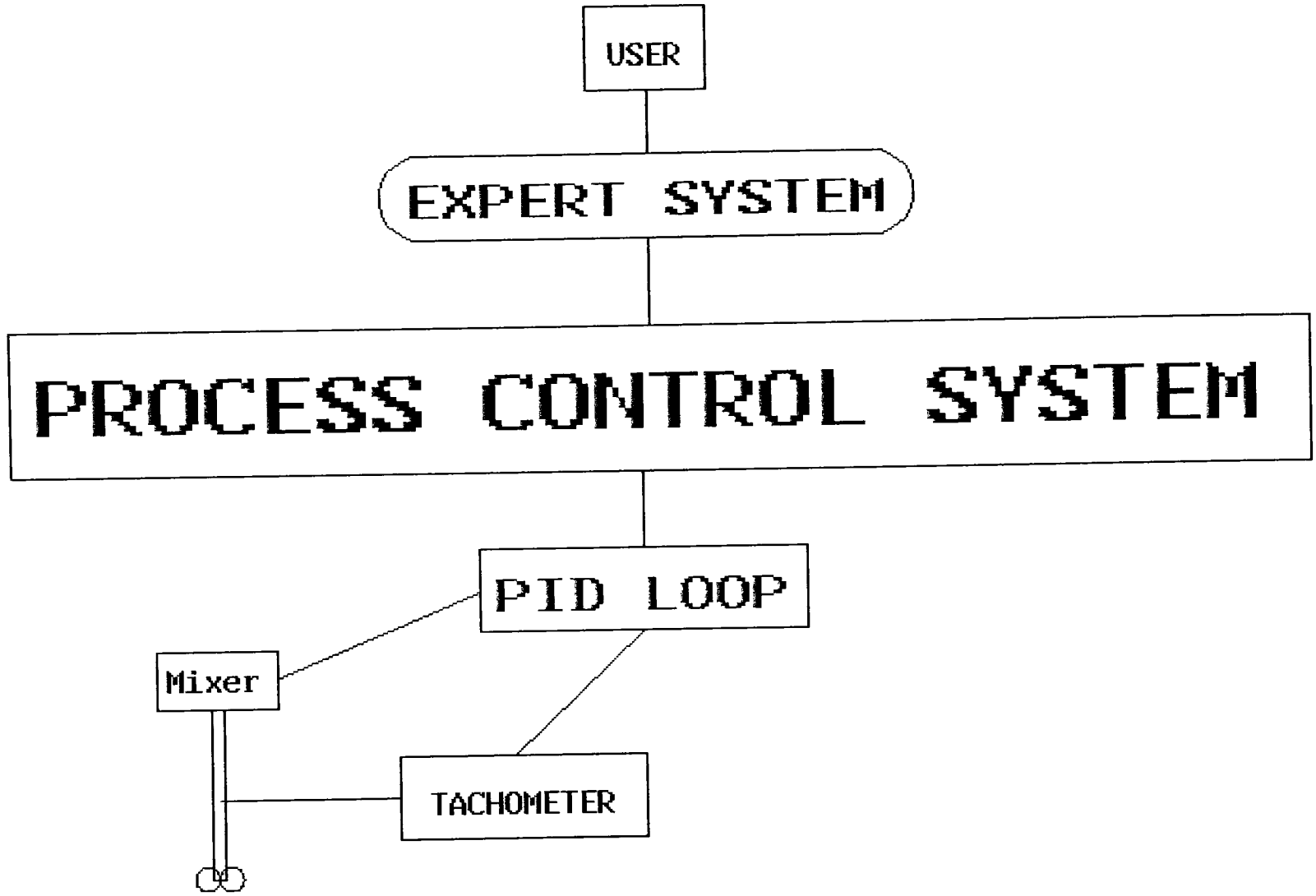

USER
EXPERT SYSTEM
PROCESS CONTROL SYSTEM
PID LOOP
Mixer
TACHOMETER

FIGURE 1.2

CHAPTER 2

BACKGROUND

2.1 PROBLEM DEFINITION

The automated manufacturing process being studied in this thesis is run by a process control system and supported by a team of engineers and technicians. The team is responsible for maintaining the equipment within specifications, where the accuracy and precision of the critical process parameters is known. However, the process has evolved to a level that makes it difficult to troubleshoot. An expert system was built to help reduce the time to troubleshoot and to recall the solutions to some of the infrequent problems.

The process control system is the automation of a batch chemical mixing process using a digital control scheme. The system includes equipment on the manufacturing line (e.g. kettles, valves, pumps), a digital interface (e.g. Analog to Digital, Digital to Analog, Discrete Input and Output), a small minicomputer for supervisory control, a programmable logic controller for process control, and a super minicomputer for recipe creation and data analysis.

The complexity of the application software and hardware requires several support groups to become involved at different levels in the troubleshooting process (See Figure 2.1). When a problem arises, a troubleshooter is called at any time of the day or night to diagnosis the problem far enough to determine which support group needs to get involved. This person is also responsible for insuring that the problem is truly fixed after each support group is finished. For a complex repair or a repeat problem, the troubleshooter may have to put in long hours at any time of day or night.

A human troubleshooter approach has many drawbacks. The production area loses time trying to locate and wait for a troubleshooter to arrive; during off-shifts and after long hours, the troubleshooter is not thinking as clearly; the production facility is vulnerable to the loss of a troubleshooter; different troubleshooters have different levels of expertise; the more infrequent the problem, the longer it takes for a diagnosis; time spent troubleshooting is taken from the troubleshooter's other responsibilities; and the cost of a human troubleshooter increases with the complexity of the process.

From each support group's perspective, it would be beneficial to replace the troubleshooter with a machine that could be run by a production operator. If a user-friendly expert system could be developed to meet this need, it would greatly reduce the production facility's downtime and risks, while improving its quality. This approach is consistent with the concept of "world-class manufacturing", where zero-defect quality levels and "just-in-time"

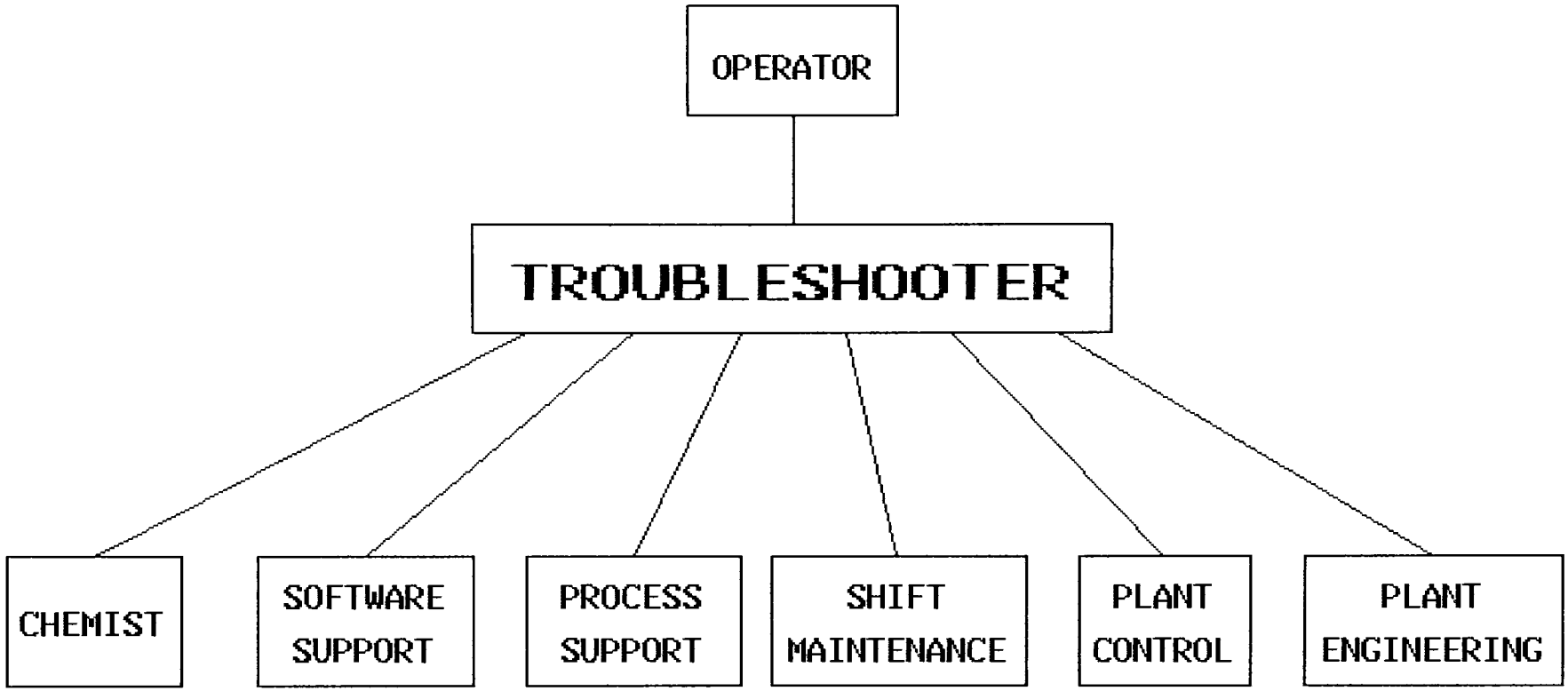

OPERATOR
TROUBLESHOOTER
CHEMIST
SOFTWARE SUPPORT
PROCESS SUPPORT
SHIFT MAINTENANCE
PLANT CONTROL
PLANT ENGINEERING

FIGURE 2.1

production strategies are key components.

The expert system should answer the basic question: Are there known problems with the application and if so how does one fix them and/or who does one call to have them fixed.

2.2 REQUIREMENTS

A standalone expert system with a significant level of knowledge would ask the user to enter too many responses to be useful. Even during the early prototyping stages for this project, it became evident that the user input was excessive. Thus from the user's perspective (the production operator), the expert system would be useful only if a large portion of its input were automated.

In order to automate the input to the expert system, a communications link was established between the expert system and the process control system (See Figure 3.1). Due to the fact that the expert system supports a 24 hour manufacturing line with a tight schedule, the communications link must be available for use at any time and must be reliable while the link is established. For this reason the only viable communications link was DecNet on Ethernet to the DEC PDP 11 process control system.

Since this is the first expert system project at this manufacturing site, a large expenditure on expert system specific software and hardware is not feasible. This project is a proving ground for future expert system projects, but until then the purchase of additional software and hardware must be kept to a minimum. Thus,

the expert system software should run on existing hardware, a VAX/VMS system, with access to the process control system via the DecNet link. In addition, the cost and system overhead prohibited the use of large expert system languages (e.g. KEE and GURU).

2.3 RELATED EXPERT SYSTEM APPLICATIONS

The FALCON project is a joint effort by the University of Delaware, E. I. DuPont, and Foxboro Inc. to develop an expert system for fault analysis of an automated chemical process. Initially, they built the expert system using Lisp on a UNIX based system and after conversion to Common Lisp, they ported it to a VAX/VMS system where it could communicate with the process control system. The inference engine is forward chaining engine; it collects the facts up front and then chains through about 650 rules to find a conclusion.

To fully develop and test the inference engine and the knowledge base, they spent a major portion of their time developing and using a software simulator. Thus, they could induce a fault and validate the system.

The process control system supplies the expert system with short term trend data every 15 seconds. The expert system converts quantitative data to qualitative data, validates sensor readings, and checks process control loops to decide if there is an equipment or control fault.

This system was designed to perform the following functions:

- o Data validation
- o A comparison between process theory and the actual process

The quantitative data is checked against tolerance limits, but it is also checked against normal process variability. For example, if a sensor is supplying very low (or no) variability in the data, then it can not be believed (the sensor may be stuck at one value).

After the data is validated, it is compared to the theoretical results that should be present if the process is functioning correctly. The team characterized the chemical reactions (mass balance) to describe mathematically how the sum of the input equals the sum of the outputs. A fault is detected if this comparison fails.

A presentation of this project [Rowi 86] emphasized the need to validate the data and to employ process theory and heuristics. Since the entire system was written in Lisp, the knowledge and the control information were intermixed. The team would have preferred being able to isolate the two entities.

2.4 EXPERT SYSTEM SHELL SELECTION

Before developing the knowledge base, an expert system shell or language had to be chosen to meet the following criteria:

- o Able to communicate with the process control system (Directly or via a network link)
- o Able to query the expert system tool for debugging and verification purposes.

- o Able to run on existing hardware (DEC VAX or PDP computers)
- o Provide access to existing databases (Directly or via a network link)
- o Able to isolate the control from the knowledge base
- o Provide a compiler oriented knowledge base for speed
- o Provide an inference engine, explanation facilities, and terminal I/O included if possible
- o Allow a mixture of terminal, database, and external routine input to the expert system without excessive overhead (Subroutine call instead of a process creation)
- o Support fast knowledge base prototyping
- o Allow automated test schemes (simulated input data)
- o Include an object-oriented representation

After a literature search (See the Bibliography) for comparisons of existing shells and languages, six shells and three languages were obtained and tested. The IBM PC based shells were not chosen because an adequate network interface was not available to the DEC PDP process control system. The larger expert system tools (KEE, S.1, and ART) were not feasible because they required their own unique system, they don't adapt to continuous monitoring because of periodic garbage collection, and they didn't provide a clean interface to existing DEC hardware. The VAX based languages (e.g. GURU) were not cost effective and the general programming languages (Lisp, Prolog, and Forth) were not selected because of the large amount of effort required to create and validate an inference engine and the user facilities (e.g. screen displays and explanation facilities), and to isolate the program control from the knowledge base. The VAX/VMS

version of Insight2+ [Insi 87] was initially selected for the following reasons:

- o The company was willing to modify the new VMS based product to meet my most important needs. (e.g. automated testing capability)
- o The product was affordable for a plant site just being introduced to expert systems.
- o The product was currently available and met the major requirements listing above.
- o The company's time line for product improvements aligned with the timing of the thesis.

With the fast pace that expert system shells and languages are being developed, one must take a snapshot of the current offerings to make decisions about the hardware and software that will be used. Throughout the four month product evaluation of Insight2+, there were many new software and hardware releases (e.g. the release of Insight2+ and EXSYS for the VAX). Since the evaluation, other major advancements have occurred (e.g. KEE is now available on VAXStations and communication is possible between VAX computers and AI workstations). Thus, the decisions were made with the intention of porting the application to other software or hardware if needed.

2.5 INSIGHT2+ PRODUCT DESCRIPTION

2.5.1 Product Overview

Insight2+ is a backward chaining, rule-based expert system shell that was developed by Level Five Research for use on the IBM PC. It is written in Pascal and uses a compiled knowledge base, which

includes access to external user routines and PC databases. Recognizing the need to communicate with an external environment, Level Five Research decided to port the product to a popular minicomputer, DEC's VAX family of computers. This allows the expert system to access distributed company databases and applications.

The inference engine is a basic implementation of backward chaining with a combination of fuzzy logic and boolean algebra. From an ordered outline of goals and sub-goals, the engine selects the next goal and backward chains through the rule base in an attempt to satisfy the goal. The engine continues down the goal outline as each goal is proven NOT to be true. The engine stops with a conclusion when a major goal is true. When confidence is enabled, each conclusion in a rule has a confidence level (from 0 to 100%) assigned to it. During backward chaining, the engine chooses the rule whose conclusion will yield the highest confidence. When a rule is fired, each fact is assigned a confidence level (e.g. the user is prompted to supply the confidence in each of their answers). The rule's final confidence is the product of each fact's confidence for antecedents that are ANDed (or the highest confidence for facts that are ORed), times the conclusion's confidence. The conclusion is true if it exceeds a specific (programmable) threshold level. For rules with equal conclusion confidence values, the engine selects the rule in the order it is found in the source knowledge base file, thus it does not evaluate the shortest path or least number of queries.

Insight2+ provides the following features:

- Fuzzy logic (confidence values on facts and conclusions)
- Multiple goal analysis through a goal outline
- Explanation facilities
- Procedural and logical rules
- External program activation (subprocess creation)
- Two external data passing mechanisms (via data files and/or VMS mailboxes)
- View, record and playback, and modification of the dynamic state of the reasoning process
- A production rule language (PRL) that is compiled
- Menu-driven user interface
- Knowledge base chaining and global facts
- Optional (and controllable) user goal selection
- Database access
- Boolean, numeric, string, and object fact types

Insight2+ does not currently provide the following features that were considered important for this project:

- An environment that easily provides for batch mode testing and full simulation.
- An object-oriented representation of knowledge.
- A complete set of debugging aids.
- An efficient interface to user-written code and external data.
- A fully functioning tool that owns up to the claims in its documentation.

2.5.2 Production Rule Language (PRL)

Insight2+ compiles a knowledge base from a standard text file into a tokenized file to increase the execution speed of the expert system. The uncompiled format is called the Production Rule Language or PRL. PRL includes the following types of commands, which must appear in the stated order:

1) Title display
2) Local and global fact declaration and initialization
3) Enabling and disabling of specific features
   (e.g. prompt the user to select a goal to pursue)
4) Goal outline definition
5) Production and Procedural rule definitions
6) User input screen formats
7) Display and report formats
8) Conclusion display formats
9) Explanation displays

The sample PRL source cod, on the following pages is a set of fragments from a terminal diagnosis expert system to demonstrate some of the PRL commands.

```
!  { Knowledge base title (displayed on each screen) and the title page }

TITLE Terminal Diagnosis Advisor DISPLAY

   T e r m i n a l   D i a g n o s i s   A d v i s o r

                  :
                  :

   Press RETURN or ENTER to start the advisor

! { Enable/Disable features }
!
THRESHOLD = 60          ! Establish the minimum confidence level that
                        ! conclusion can have and still be TRUE
CONFIDENCE ON           ! Enable confidence evaluation (vs. discreet)
GOALSELECT OFF          ! Do not allow the user to select a goal,
                        ! let the program try all goals.
```

Figure 2.2

Figure 2.2 shows a knowledge base heading. The title "Terminal Diagnosis Advisor" is displayed at the top of every screen when the expert system is executed. The text between the DISPLAY command and the exclamation mark is displayed as the introductory screen when the expert system is first started. The THRESHOLD command sets 60% as the minimum confidence level that a conclusion can have and still result in an affirmative conclusion. The CONFIDENCE command enables the use of confidence factors between 0 and 100%. When confidence is disabled, only factors of 0 and 100% are used. The GOALSELECT command forces the expert system to consider the goals in the goal outline from top to bottom until a conclusion is met. When goal selection is enabled, the user is presented with a list of goals to chose from. The user has the option of selecting a single goal to pursue or

letting the expert system pursue all of them as if goal selection were disabled.

```
! {Goal Selection Outline}

1          power to the screen
1.1          terminal can communicate
1.2          setup problem
1.3          only this port is bad
1.3.1          terminal should be replaced
1.3.2          port or wire is bad
1.3.2.1          wire is ok
1.3.2.1.1          wired incorrectly or bad connectors
          :
1.4          console terminal is ok
1.4.1          terminal port may be busy
1.4.1.1          terminal port is busy
1.4.1.1.1          terminal port was busy
1.4.2          priority deadlock problem
2          power to terminal problem
          :
```

Figure 2.3

The goal selection outline in Figure 2.3 is a hierarchy of possible goals to pursue. This provides a structure for the inference engine to follow as it backward chains through the knowledge base. When goal selection is disabled, the inference engine will evaluate goals one at a time from the top of the goal outline to the bottom. This is a depth first search of the outline. The search stops when a conclusion is drawn for a major goal (e.g. 1 or 2, but not 1.1 or 1.3.2.1).

The knowledge engineer creates the outline to steer the inference engine through the possible conclusions. The goals can be ordered according to the likelihood of occurrence, as pre-requisites to other

goals (e.g. a goal may fire procedural rules to collect data), or with increasing complexity. In this example, the major goals are organized by the likelihood of occurrence, and the sub-goals are ordered by pre-requisites and complexity. For example, checking the power switch (1. power to the screen) is easier than checking the continuity of the wire (1.3.2.1 wire is ok).

```
! { Rule Set }

RULE to determine if the terminal is working
IF terminal responds                                          ! 1.1
THEN terminal can communicate
AND CHAIN emulopr                        ! Chain to next knowledge base

RULE set if no scroll is enabled         ! Production rule
IF no scroll is enabled
OR character size is incorrect
OR parity is incorrect                                        ! 1.2
OR keyboard connection error
THEN setup problem  CF(80)
AND DISPLAY describe setup problem       ! Conclusion display
ELSE NOT setup problem

RULE get port info from the computer     ! Procedural rule
ACTIVATE PORT_INFO.EXE
SEND wire number
RETURN setup status
RETURN baud rate
RETURN terminal port name
THEN port info retrieved
```

Figure 2.4

A portion of an Insight2+ rule set is shown in Figure 2.4. Preceding sections of the knowledge base described the programming structure similar to convention programming headings, but this section states the relationships within the knowledge domain. This sample rule set illustrates a subset of the types of rules that are

available.

The CHAIN command will cause the inference engine to abandon this knowledge base and execute the "emulopr" knowledge base. Thus, the knowledge can be organized into subdivisions. Information can also be passed between chained knowledge bases. The "set if no scroll enabled" rule has been assigned a confidence level of 70%. If the rule is affirmative, then there is a 70% confidence that this is correct. The DISPLAY command will force a specific textual description labeled "describe setup problem" to be displayed to the user when a setup problem is concluded. The "get port info from the computer" rule is a procedural rule that tells the inference engine to activate the program called PORTINFO.EXE to obtain information (e.g. database or real-time information) from a source external to the expert system tool. The expert system supplies the wire number to the external routine, which passes back the terminal's setup status, baud rate, and port name.

The goal outline is tied to the rule set by conclusions in rules that match the names of goals. For example, the first rule's conclusion "terminal can communicate" matches goal 1.1. Thus, when the inference engine selects goal 1.1, it backward chains to fire the rule named "to determine if the terminal is working".

```
! { Textual displays for terminal Queries }

TEXT voltage present                                          ! 1.3.2

    Measuring Voltage at the Terminal Connector

   o  Get a volt/ohm (VOM) meter
   o  Gently remove the connector at the back of the terminal.
   o  Set the VOM to read 0 - 15 volts DC.

   For 20 mA current loop (Mate-N-Lock 8 pin connector; 4 conductor wire)
      o  Hold connector with the protruding guide up (left side = pin 1)
      o  Measure voltage across pins 7 and 3 (transmit)
      o  Measure voltage across pins 2 and 5 (receive)

   For RS232 (25 pin connector; 9 conductor wire)
      o  Get a small flat blade screwdriver to remove the connector
      o  Measure the voltage across pins 1 and 2 (transmit)
      o  Measure the voltage across pins 1 and 3 (receive)
      o  Measure the voltage at pins 5,6,8

   Do you detect a positive or negative DC voltage?
```

Figure 2.5

The TEXT commands, illustrated in Figure 2.5, are used by the knowledge engineer to carefully word the questions that the user will be asked. When a rule fires, the antecedent can contain facts with the same name as the TEXT titles. When the inference engine finds the "voltage present" fact for the first time, it will query the user with the text in this TEXT command.

```
! { Conclusion screens }

DISPLAY describe bad wire

      Conclusion:   Bad wire # [wire number]

  A bad wire is a very rare occurrence to date, so please consider
  the following possibilities:

     o  A solid connection was not possible during testing.

     o  Broken wire where the insulation was cut away.

     o  Cold solder joint or broken wire under the shrink tape (RS232)

     o  Poor connection in the crimped pins (20 mA)

     o  Terminal wired to a port that is not connected to a controller.
```

Figure 2.6

When a conclusion is reached, the inference engine can display the conclusion name (e.g. bad wire = TRUE) or a custom display can be defined to display detailed descriptions (See Figure 2.6). With custom displays, an engineer can state the problem and detail the solutions for future prevention. In the example above, the knowledge base stopped short of finding the exact cause, but instead outlined the possible causes.

These textual displays can display the value of specific facts. In the example above, the wire number would be substituted for the [wire number] text in the display.

```
! { Explanation facility }

EXPAND terminal communications setup                          ! 1.2

       Terminal Setup Characteristics

    Refer to the setup standards at your facility, but
    here is a list of the important characteristics:

                          Typical values

    Transmit speed              9600 baud
    Receive  speed        always same as transmit speed
    Number of Stop bits         1 bit
    Character size              8 bits   or   7 bits
    Parity                      None          Space
    Online/Offline              Online

    Terminal Mode               VT200 for VT200
                                no such setting for VT100

    Electronics                 20 mA port

END
```

Figure 2.7

The EXPAND command is an explanation facility, which is illustrated in Figure 2.7. The text in the EXPAND commands are not displayed to the user unless he/she presses the EXPLAIN function key. The title in the EXPAND command "terminal communications setup" is the name of a corresponding fact or conclusion.

2.5.3 Required Improvements

To maximize the usefulness of Insight2+ for this application, a few changes were recommended to Insight's developers, Level Five Research.

- Include forward chaining mechanisms.
- Provide the ability to execute INSIGHT2+ in a batch command procedure
- Provide mailbox communications with a detached process.
- Provide command line compilation of knowledge bases.

With the use of automated retrieval of a group of facts and the hope that the application will eventually run continuously, forward chaining mechanisms would be more efficient. In order to automate the validation of the software and the knowledge base, Insight2+ must be executable in a batch procedure. This would improve the reliability of the testing and greatly decrease the manual labor required to enter test information. The mailbox communications directly to a detached process would reduce the overhead of firing up a subprocess, which would retrieve the information from Insight2+ and passes it to the detached process. Command line compilation would allow multiple knowledge bases to be kept up-to-date across many installation sites. This is important because the expert system will be distributed to multiple manufacturing facilities.

An object oriented representation is preferred to reduce the rule base by describing static and dynamic information as classes and objects and writing generalized rules about the classes and objects. INSIGHT2+ does not currently offer object representation.

2.6 THE SELECTION OF A NEW EXPERT SYSTEM SHELL

After 4 months of testing and debugging INSIGHT2+, it was decided that the product and the company were not living up to their

advertising and their promises. Without a viable expert system tool for the VAX, the thesis work shifted for 5 months on the design and the initial coding of a custom built expert system tool written in PASCAL on the VAX. With money available in the budget at the end of the year and the availability of a Macintosh SE computer, it was decided to purchase the NEXPERT expert system tool. A development version was purchased for the Macintosh and a runtime only version of NEXPERT was purchased for the VAX. NEXPERT became the expert system tool that was used to develop and implement the knowledge base.

2.7 NEXPERT PRODUCT DESCRIPTION

NEXPERT [Nexp 88] is a hybrid expert system shell that is a product of Neuron Data. It is written in C and can be ported to a number of popular computers (e.g. IBM PC, Macintosh, VAX, SUN, and HP). Knowledge bases can also be ported from one type of computer to another (e.g. a knowledge base can be developed on the Macintosh and executed on the VAX). A hybrid expert system shell integrates production rules with an object-oriented representation. The rules contain a premise section, a conclusion, and an action section; the actions are executed if a rule fires.

The object representation allows for classes, sub-classes, objects, and sub-objects (See Figure 2.8). Each of these can have property slots. The property slots and/or their values can be inherited up and/or down an object structure. An object's or class's property slots can have unique meta-slots associated with them to define initial or runtime values and/or determine the source of the

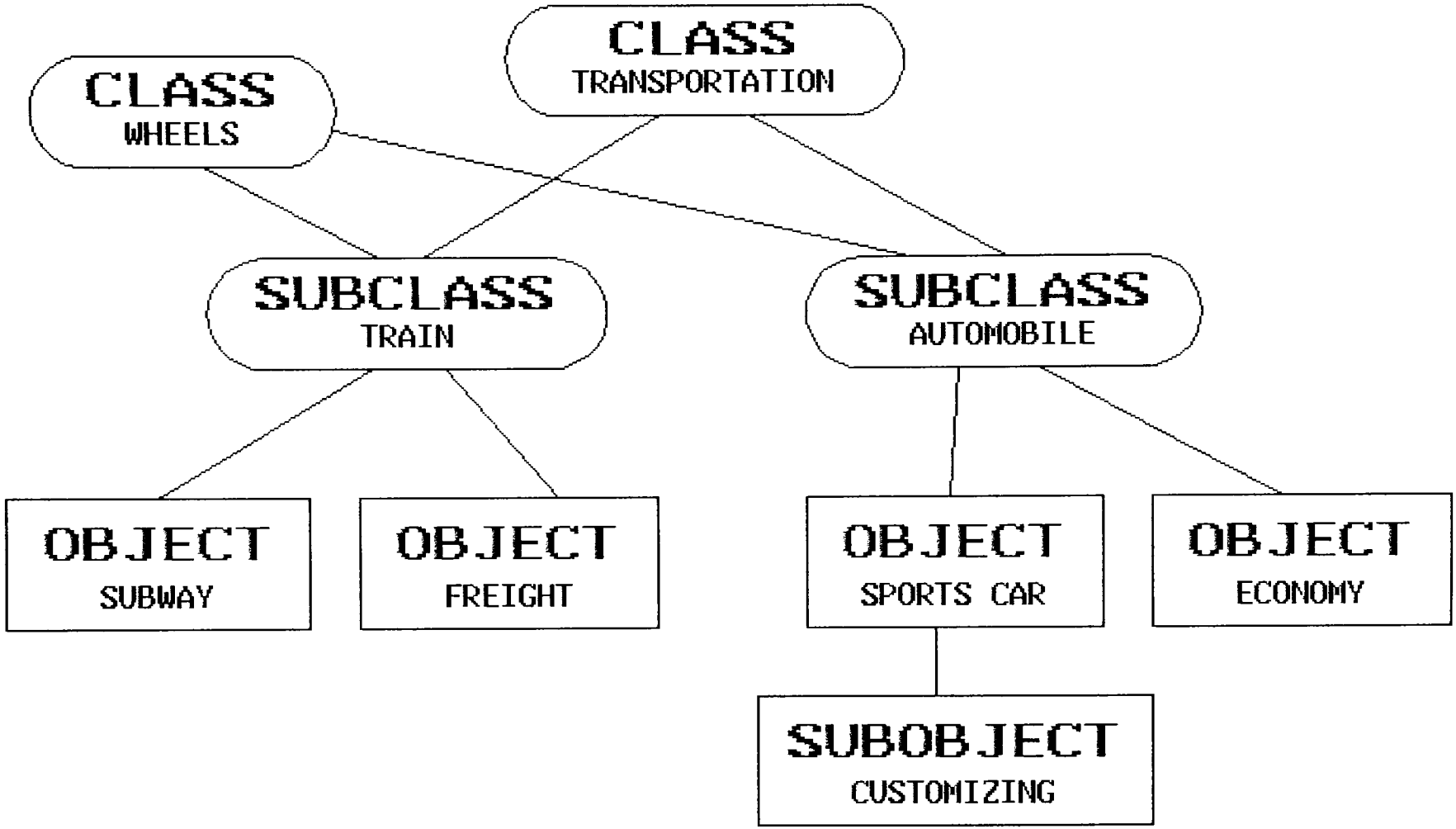

CLASS
TRANSPORTATION
CLASS
WHEELS
SUBCLASS
TRAIN
SUBCLASS
AUTOMOBILE
OBJECT
SUBWAY
OBJECT
FREIGHT
OBJECT
SPORTS CAR
OBJECT
ECONOMY
SUBOBJECT
CUSTOMIZING

FIGURE 2.8

data value (e.g. by asking the user through an external subroutine call or through a database query). When property slot values must be inherited and there are multiple sources available, NEXPERT will follow the inheritance strategy (e.g. class first, object first, breadth first, or depth first) and select the value with the highest inheritance category. The knowledge engineer can set the inheritance strategy and category for each object or class property slot. The relationship between classes and objects can be static, where the knowledge engineer defines them at compile time or they can be established dynamically at runtime. An object can be dynamically created or deleted by a rule or from a subroutine call by the application. Also, objects can be linked or unlinked from a class or parent object.

NEXPERT can follow a forward chaining strategy, a backward chaining strategy, or a mixture of both. This is referred to as an augmented rule control structure. NEXPERT is event-driven, thus if data is "volunteered" into NEXPERT from any number of possible sources, NEXPERT will evaluate all the rules that use this data in its premise. If a rule is "suggested" by an external source (e.g. the user, an application routine, or an "if-change" clause from a property slot), NEXPERT will backward chain in an attempt to fire this rule. During each NEXPERT inference pass, the inference engine gathers a list of all rules whose data has changed or were suggested since the last pass. The list of rules will be sorted by their inference category, which is set by the knowledge engineer to steer the evaluation in a path that is logical to the user and the problem.

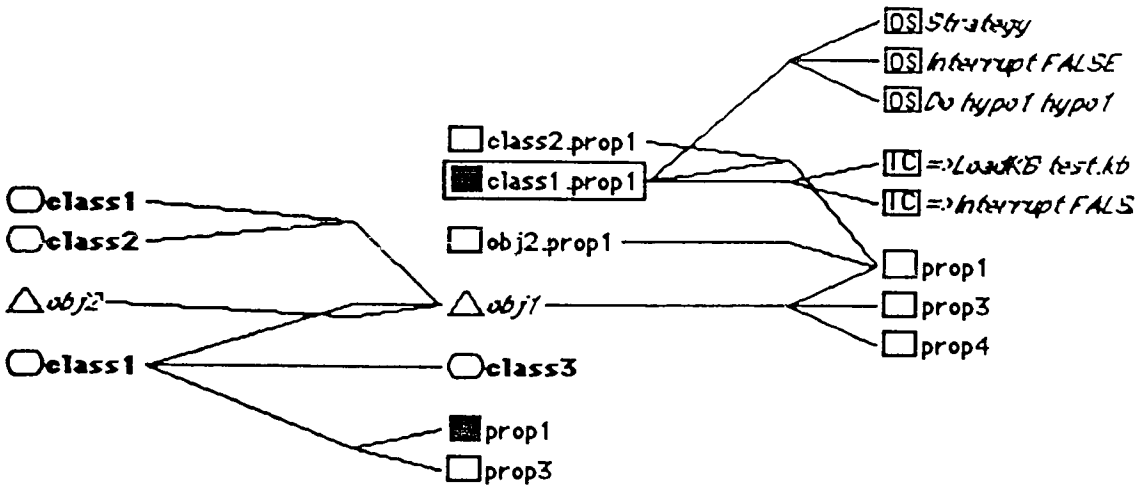
NEXPERT allows a knowledge engineer to design a knowledge base in modules or knowledge islands. These islands can be in separate knowledge base files that are loaded into memory as needed by an application program or by a rule. If a knowledge base is loaded after another knowledge base file, the two will merge. If the same class or object is defined with different properties, then the resulting class or object contains all of the properties. Thus different knowledge bases can contribute to the knowledge about a class or an object. For example one knowledge base may investigate the hardware of a computer, while another investigates the software. From the inferencing standpoint of knowledge islands, rules can be indirectly related between islands. The conclusion in one rule may not be present in the premise of another rule in a different knowledge island, but one rule can have a "weak link" to the other rule. Thus, if the first rule is investigated, the second rule can be investigated. The knowledge engineer can determine when weak links will be followed using the context editor at compile time or dynamically from a rule.

The VAX version of NEXPERT is a shareable library that allows the shell to be linked directly with application software. Thus the NEXPERT inference engine can be embedded within an application to the point where it is not detectable. This linkage allows the inference engine to communicate directly with the application through subroutine calls. The application can pass information into NEXPERT, a rule can execute a subroutine when it is being evaluated or has been fired, or a rule can backward chain to request a piece of data, whose source is the execution of a subroutine.

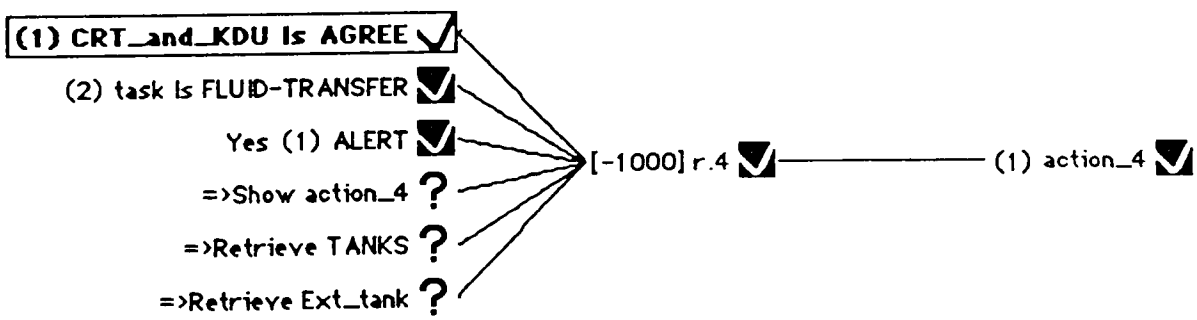
NEXPERT has several levels of communication with the outside world, which is typically a user. These communications are carried out through handler routines that can be customized for an application. Thus, the knowledge engineer can choose to pass the communications on the user (e.g. a standalone expert system) or have the application software complete the communications automatically. There are handlers to display text or graphic files, to flag an abnormal state or error, to ask a question, or to execute a custom application subroutine. The handlers allow the knowledge engineer to control the amount of user interaction and to add custom features (e.g. value substitution in text displays).

The ability to control the user interface provides the opportunity to define an automated test environment. If the user input can be supplied by a file or multiple files, then a fully automated test scheme can be implemented. Thus, each rule can be tested when it is defined and retested when the knowledge base is modified. Most expert system shells are tightly coupled to a user's terminal, which requires all testing to come from the terminal. For large knowledge bases, this is a tedious and unreliable testing method.

The area, which Neuron Data has put a lot of thought into, is the debugging and reporting capabilities in NEXPERT. NEXPERT can graphically display a rule or object network to show the relationships between rules or between objects and classes (See Figure 2.9). Icons and colors are used to graphically show the current state of the inferencing. One can easily determine if a rule has been or is



Object Network



Rule Network

Figure 2.9

currently being evaluated, the state of each rule's premise, and the linkage between rules. These features are important and implemented nicely, but they require a computer with screen "Windows" (e.g. VAX or SUN workstation, a Macintosh or an IBM PC, but not a dumb terminal). Thus, an application written to run on a standard DEC terminal and VAX computer can not directly take advantage of this. However, the latest version of NEXPERT (version 1.1) has a journaling mechanism that should allow an expert system executed and journaled on the VAX to be played back and analyzed on a Macintosh.

As with conventional programming, expert system programming will migrate towards structured programming techniques. NEXPERT simplified the rule format by leaving out an ELSE segment and the OR operator, which are important components of conventional structured programming. Without these features, alternate rules must be written to accomplish the same function. The number of rules written could be reduced and the clarity of the knowledge could be improved if the ELSE segment and the OR operator are implemented.

NEXPERT fit this application's expert system requirements nicely, because of its ability to communication directly with the application and the flexibility available to the knowledge engineer in the implementation of the expert system. Thus, data can be passed to the inference engine in a timely fashion, a batch mode test scheme can be instituted, and the graphic rule and object networks help analyze and convey the contents of the knowledge base.

CHAPTER 3

IMPLEMENTATION

3.1 OVERVIEW OF THE SYSTEM DESIGN

This is a layered software system ranging from the process control hardware up to the inference engine. It uses software built in-house and purchased; it crosses multiple languages and computers; and it is designed to be run at multiple manufacturing facilities.

A model of the software system can be seen in Figure 3.1. When the user suspects a problem with the process control system, he or she will invoke the Emulsion Make Diagnosis knowledge base using the expert system routine. The expert system establishes a link with the process control system to assess the state of process.

If the process control software is aware of any errors, it will pass this information back to the expert system for analysis. This type of error has fairly concrete solutions, which the expert system can recommend to the user without a complex set of rules. For example, the process control system may report that there is insufficient disk space, and the expert system will recommend that a

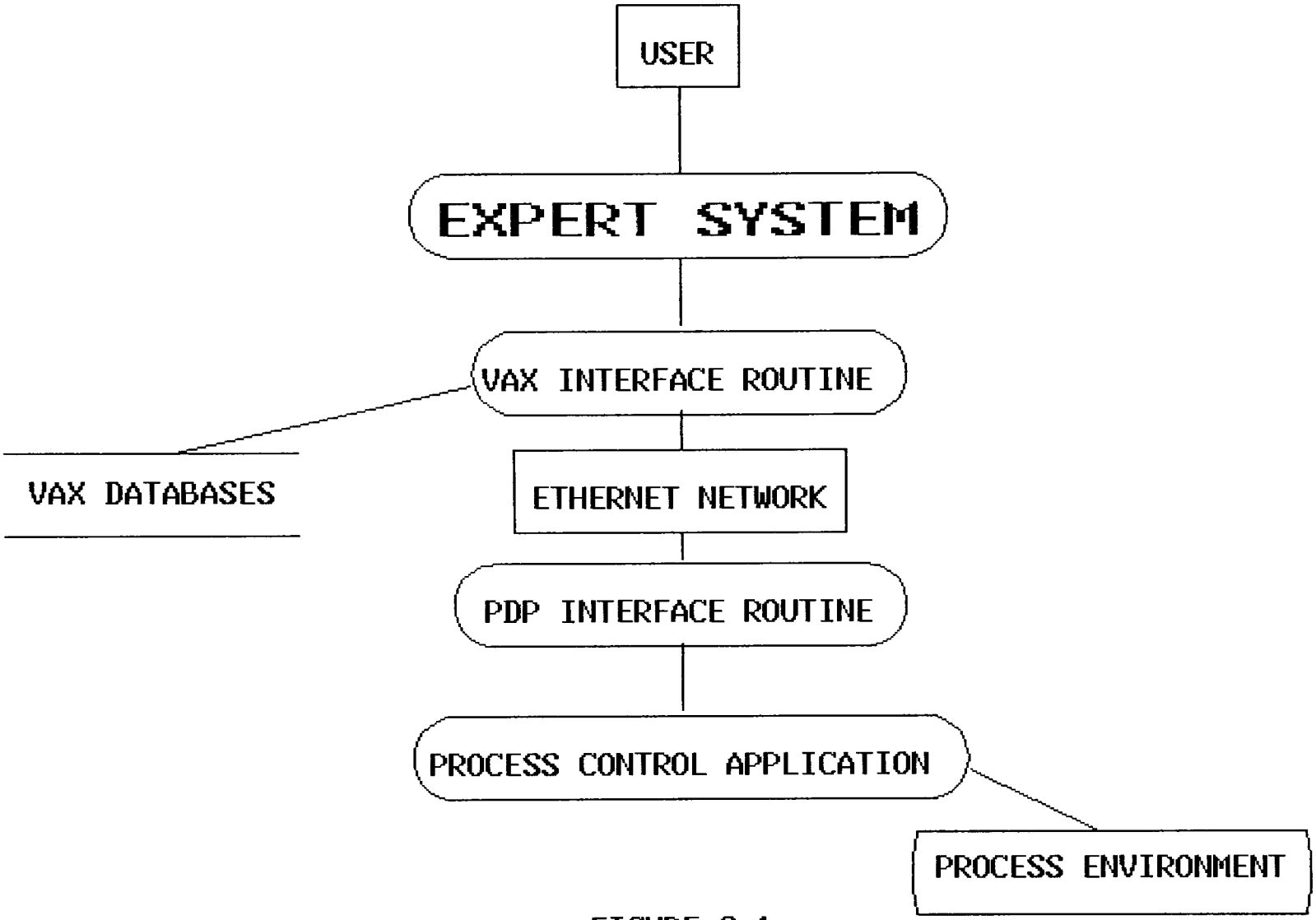

USER
EXPERT SYSTEM
VAX INTERFACE ROUTINE
VAX DATABASES
ETHERNET NETWORK
PDP INTERFACE ROUTINE
PROCESS CONTROL APPLICATION
PROCESS ENVIRONMENT

FIGURE 3.1

member of the computer operations group be contacted to recover the disk space.

After verifying that the process control system's vital components are functional, the expert system will evaluate all process in an order that is pre-defined by the knowledge engineers to test for the most significant or frequent problems first.

A rule will be investigated by querying the process control system directly for as much information as possible. If the process control system cannot answer a question, then the user will be asked. This will continue until all aspects of the process have been investigated using automated data acquisition or until the user decides to exit. At this point the expert system will ask the user for any information about a specific area of the process that the user wants the expert system to consider.

3.2 DETAILED SYSTEM DESIGN

3.2.1 Expert System To Process Control System Interface

To minimize the number of user questions and to improve accuracy of the data, a network interface is used between the expert system and the process control system. A model of the interface is shown in Figure 3.2.

The expert system establishes a logical network link to the process control computer via the DecNet protocol (DDCMP and Ethernet). This link will remain active while the expert system is running. To

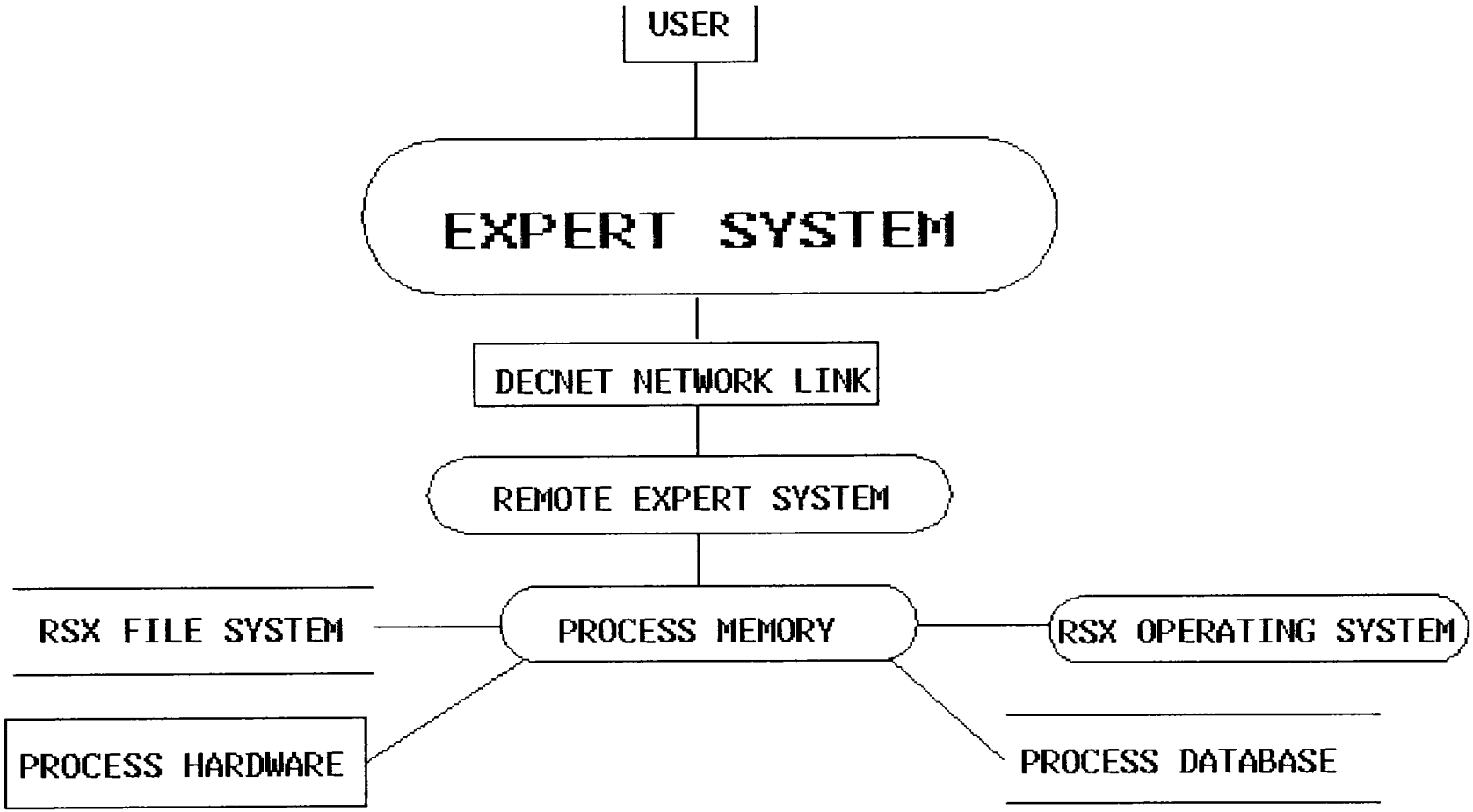

USER
EXPERT SYSTEM
DECNET NETWORK LINK
REMOTE EXPERT SYSTEM
RSX FILE SYSTEM
PROCESS MEMORY
RSX OPERATING SYSTEM
PROCESS HARDWARE
PROCESS DATABASE

FIGURE 3.2

create a logical network link, the VAX computer will initiate a specific process (task) on the PDP computer and establish a bi-directional synchronous message buffer between the two processes.

The PDP-specific process is called the Remote Expert System. It will remain active while the VAX process is active. The Remote Expert System will retrieve information from the process control environment and send it to the expert system via the network interface.

The automated data acquisition routines are algorithmic units that schedule the collection of information from the process control system and convert the raw data from the process control system into singular quantitative or qualitative information, which is useful to the expert system. Since the expert system may be in use for a period of time, the state of the process control system might change. Thus the Remote Expert System will take a snapshot of the process environment within a second or two so that the expert system inferencing speed does not influence the accuracy of the results.

The Remote Expert System is a facility-specific routine that is capable of accessing any piece of information within the Process Control Data Structures. This routine will convert data into units that are facility-independent. For example, one facility may use flowrates that are 50 times higher than another facility, so the Remote Expert System will convert the raw flowrate to percent of a maximum flowrate before passing it to the expert system.

3.2.2 Process Control System

The Process Control System is the integration of multiple components of hardware and software that are used to monitor and control the chemical batch manufacturing process.

The VAX computer is used by the support teams to perform higher level support functions. This isolates them from inadvertently affecting the manufacturing process, which is run on the PDP computer. The VAX is used to create product recipes, to test new software and recipes under simulation, to archive historical data, and to perform data analysis, graphing, and reporting. The product chemist creates a recipe on the VAX, which is approved by a process engineer, who releases it to the PDP computer. The expert system is run on this VAX using either simulation or actual process data. The network connected to the VAX allows this expert system to be used to diagnose all of the sites that use this process control system.

The PDP computer handles the interaction between the line operator and the equipment through automated recipe software that prompts the user and performs supervisory control over the equipment. The Programmable Controller receives supervisory commands from the PDP, which it executes continuously until the PDP supersedes it or the controller detects an error. The PDP orchestrates the process, while the Programmable Controller accurately performs each function as an independent entity.

3.2.3 Process Control Data Structures

The Process Control Data Structures encompass all the data representations on each component of the Process Control System. The Interface software is concerned with knowing how the data is represented and how to access it. The expert system expects the interface routines to handle the data acquisition in whatever form it takes. This will require accessing flat data files, relational databases, operating system and file system data structures, shared memory, and reading the state of the hardware.

3.3 DETAILED KNOWLEDGE BASE DESIGN

With all of the tools for retrieving and evaluating facts about the process, it is time to describe exactly what the tools will do. This is a description of the knowledge domain, the set of conclusions the expert system will attempt to draw from available facts. The set of conclusions can be represented with a decision tree, which can become very large as more knowledge is acquired and the expert system is refined.

To provide a useful tool early in the development stages and to learn early about the problems involved in implementing the expert system, the initial expert system should be developed in a breadth-first fashion. Thus, initially the expert system will be developed to steer the user in the right direction without drawing any exact solutions about a specific problem. For example, it may state that the terminal port controller is malfunctioning without an

exhaustive search to find that the wire on pin 2 is broken. After the first development pass, each branch of the decision tree can be extended to include more detailed investigations and more specific solutions.

From discussions with the team of experts, the following six areas were considered the most important to begin solving:

1) Major computer hardware problems
   (e.g. CPU, disks, network)

2) Process control configuration problems
   (e.g. missing components)

3) Process control software errors
   (e.g. software development error)

4) Operator errors
   (e.g. doing something out-of-sequence)

5) Intermittent or Marginal Equipment Problems

6) Process is out-of-specifications

This is a diversification of knowledge to gain experience from different levels of implementation complexity. Some areas have an exact solution based on single and multiple measurements; some require long paths with many options to multiple solutions; others incorporate multiple levels of knowledge (meta-knowledge) and confidence factors to draw a conclusion; and some areas use statistical analysis of dynamic and volatile process data (data which is valid for only a short period of time).

3.3.1 Major Computer Hardware Problems

Conclusions about computer hardware problems will be exact

solutions based on reliable information. This area of conclusions is included because the rest of the conclusions cannot be accurately drawn until we are sure the equipment is fully functional. It is also an area that the operator knows the least about and the expert system can conclude without operator input.

The following is a list of hardware problems that the knowledge base will understand:

- o Computer network is down.
- o Process control computer has crashed.
- o A component of the process computer is malfunctioning
  - disk errors
  - memory errors
- o Programmable controller is offline.

3.3.2 Configuration Problems

In order to conclude that there is at least one configuration problem, the system must investigate a large number of missing or corrupt components, thus producing a shallow decision tree with many branches. This type of investigation is fitting for an expert system. Because the problems occur infrequently, they are typically caused by inexperienced application managers (from new or upgraded facilities), and the cause is not obvious to an operator, who is not trained to understand the inner workings of the application software. This type of problem appears elusive because it is not typically suspected, thus it is not resolved until several levels of expertise have proven that the problem is not in their area (not operator related, not process

equipment related, and not computer hardware related).

The knowledge base will investigate the following configuration problems:

- o Configuration and recipe data file, command procedure, or program is missing or corrupt.
- o Program not installed and/or not running.
- o Equipment configuration data is not loaded into memory.

3.3.3 Software Errors

With the size of the process control software application, the use of concurrent programs (up to 10), multiple levels of computers (VAX to PDP to Programmable Controller), there is a risk of a component malfunctioning. The process control application was designed to pinpoint an error as soon as it occurs and shut down the process to prevent quality and safety problems. The application does not include a detailed solution to each problem that occurs. Instead it flags the module(s) that malfunctioned and the latest error status that was detected.

The diagnosis of the error has been left for the expert system, because a given module is only programmed to know about itself. Thus, a module does not carry the specific solutions to all possible errors that may result from a lower level module (e.g. a program that controls a process loop puts the process onhold because a module three levels down detected an I/O error) or a concurrent program induced the problem (e.g. a process limit was exceeded in one program, because

the program that reads the sensor detected an I/O error). Instead, the expert system will view the entire set of errors in an application in order to find the source of a problem.

The following list describes the software errors that the expert system will interpret:

- o A positive error status (an error that can be corrected without re-programming e.g. a recipe has misspelled an equipment name).
- o Hardware errors detected by the application software (e.g. an analog input channel is malfunctioning).

3.3.4 Operator Errors

This area of solutions requires a higher level of operator interaction with the expert system than other areas. The expert system will compare the facts from the process computer with the facts from the operator's responses. These problems arise because an operator is new or has assumed he/she knew what to do and ignored the descriptions on the screen. The expert system will refresh the operator by describing the sequence of operations again with more detail. But this time, they'll pay more attention.

The following is a list of operator errors that will be detectable by the expert system:

- o Operator is not logged into the computer or the application.
- o A product recipe has not been selected or is complete.
- o Operator cannot continue with the procedure (e.g. operator doesn't know how to exit from an option).

- Operator has not completed an operation (e.g. the START-MIXER button must be pushed).

3.3.5 Intermittent Or Marginal Equipment Problems

Equipment problems have been addressed in two other areas (major problems and software detected problems), both of which relied on repeatable errors that are immediately detected by the system or application software. Intermittent problems and ones that are trending out-of-specification can only be accurately detected by statistical trend analysis from multiple sources of information. The expert system will be presented with contradicting data (e.g. the application software believes it is functioning and within limits on average, but the trend analysis and process control variance says it is degrading). Also since the control loops are designed to automatically correct for a finite set of system upsets, they may hide a problem until the equipment fails. This area will resolve contradiction with confidence factors and statistics.

The following is a list of problems using meta-knowledge to detect subtle equipment problems:

- Pump head wear.
- Degrading sensor calibrations (e.g. electrodes, flowmeters).

3.3.6 Process Is Out-of-specifications

In this area, the application software discreetly detected that

the process is out-of-specification and shut the process down, but the software does not know why. A cascading reaction could have occurred where a process limit was exceeded due to a related problem (e.g. the process control was erratic because the mixer speed was incorrectly set). The expert system is properly equipped to handle this type of problem with its backward chaining approach. Given the single conclusion that a process parameter is out-of-specification, the expert system backward chains through all the rules that relate to this problem. This offers a high payback, because it can include even the most infrequent and obscure possibilities that a human expert would overlook or forget.

The following is a list of out-of-limits conditions that the expert system will recognize:

- Primary controlling electrode aim and standard deviation (This conclusion has a large number of possible causes).
- Controlling pump variance.

3.4 APPROACH

This section will describe how the knowledge base will be developed, validated, and implemented. The development of knowledge bases (knowledge acquisition) is getting considerable attention in the area of expert systems, because it is a labor intensive process that is labeled as a bottleneck in the development of expert systems. It is difficult to extract enough information from an expert and then represent it in a knowledge base so that it is accurate and reliable.

Furthermore, this acquisition process is complicated by the contradictions between two or more experts.

A major purpose of this thesis was an evaluation of the knowledge acquisition techniques that were employed. The results and conclusions drawn from the implementation of this expert system focus on the success (or failure) of these knowledge acquisition techniques when they are used for diagnosis expert system with an automated interface to the process control application.

The following is an outline of the development approach that was used:

- Select four knowledge experts for this application and introduce them to the idea of expert systems.
- Have two experts learn NEXPERT.
- Meet with experts to identify the most important areas to be addressed.
- Interview experts separately to build a skeleton knowledge base.
- Test the knowledge base under simulation.
- Pilot test with a select group of operators.
- Have the experts and the knowledge engineer verify and expand the knowledge base as new solutions are identified.
- Develop operator oriented screen displays and explanation facilities.
- Train all operators.
- Release for general use.

Since the experts and the knowledge engineer are both process engineers and computer programmers, communication losses were reduced by having the experts load the knowledge base directly, whenever possible. The knowledge engineer acted as a resource to bring about a consensus among experts, when there was contradictory information.

To allow the experts to express their concerns and to introduce everyone to the knowledge acquisition stage, individual interviews preceded group interviews. Group interviews were treated as brainstorming sessions to draw-out subtle points and increase each expert's understanding of the project and the application.

The knowledge base was continuously tested under simulation, using a continuously enhanced test scheme. The test scheme helped to validate software and knowledge base changes and document the testing process. The simulation is a set of programs that simulate the process parameters and allow malfunctions, errors, and noise to be injected at any time. Thus, the expert system was developed "off-line" to a useful level before presenting it to the operators. The knowledge engineers and the experts used the prototype immediately to test problems that they were called in to fix. Any inaccuracies were corrected and retested using the test scheme.

Once the knowledge was accurate, the data entry, conclusion, and explanation screens were updated to become useful to the operators. The final screens were paragraphs rather than phrases. Select operators were used through the development for guidance, but a considerable time savings was gained by delaying the detailed screens

until the knowledge base was stable.

With the approval of the experts and the operators, the expert system will be released following operator training. The experts are responsible for the currency of the expert system after it is released. Beyond this thesis, it is hoped that the long term use of the expert system (and others like it), may result in the operators becoming the experts to the point where they will support sections of the knowledge base.

3.5 ISSUES

There are several techniques being used in the design of this knowledge base that the recommended practices described in the literature.

The most significant deviation from the recommended practices is the knowledge engineer serving as an expert. When the knowledge engineer is also the expert, there can be a loss of information, because the expert takes a lot for granted [Wate 86]. A separate knowledge engineer requires more clarification and can view the domain from a different perspective to form a more complete representation of the knowledge. Also, the separate knowledge engineer can concentrate on knowledge representation. In this thesis, the small size of the knowledge base and the use of multiple experts helped to minimize these factors.

The literature recommends using one major expert for the source of the knowledge [Harm 85]. Since, interviewing experts is a time intensive task, it is difficult to interview multiple experts completely. Also, multiple experts tend to contradict each other on the finer details. This thesis used one principle expert, but each expert had his/her own unique areas of expertise that was beyond the scope of the principle expert. Also the group of experts were used to intentionally challenge the knowledge from the other experts to provide a more accurate and complete knowledge base. For all other job functions, this group functions as a team, thus the team approach was expected to apply to expert system development as well.

Recommended knowledge engineering techniques warn against exposing the experts to the expert system internals [Wate 86]. When the experts are familiar with the way knowledge is represented in the expert system, their knowledge is said to be contaminated. The expert will attempt to translate knowledge to fit the expert system tool's representation scheme, while leaving out information which is not easily represented. Only half of the experts were allowed to become familiar with the expert system tool, so we could detect and correct any signs of contamination.

3.6 TIMELINE

The previous sections described the details of the work that was done to produce an expert system application. Below is a timeline that describes the timing of each milestone in the thesis. The deliverable items are shown in parentheses.

Thesis Timeline

9/86 - 12/86	Expert system language and shell evaluation. (the selected tool)
11/86	Submit a pre-proposal.
1/87 - 4/87	Feasibility study of Insight2+ on the VAX. Request required changes to the Insight2+ software.
	Prototype the interface to the process control system. (the prototype interface)
2/87	Initial discussions with experts.
3/87	Experts attend a training course on Insight2+. Expert approval of the major goals in the expert system. (a list of major goals)
	Write the thesis proposal.
4/87	Develop the interface to the process control system.
	Continue the literature search. (updated specifications and background description)
5/87 - 8/87	Write a complete test scheme to verify all aspects of the Insight2+ expert system shell. (canceled the use of Insight2+)
8/87 - 9/87	Design our own expert system shell for the VAX.
9/87 - 12/87	Develop and test the expert system compiler.
12/87	Acquire funding to purchase the NEXPERT expert system tool for the Macintosh and the VAX.
1/88	Learn NEXPERT

Write a conversion routine to allow a Macintosh knowledge base to run on a VAX system.
Test all aspects of the NEXPERT subroutine library.

2/88 - 3/88 Construct a generic debugging environment on the VAX to allow for complete simulation and verification of NEXPERT knowledge bases.

2/88 - 4/88 Interview the experts to build the knowledge base.
Test the knowledge base under simulation.
Pilot test with a select group of operators.
On-going validation of the knowledge base.
(a functional rule-set)

4/88 - 5/88 Determine that the expert system is stable.
Finalize the operator oriented screen displays.
Finalize the explanation facilities.
(a functional operator oriented knowledge base)

5/88 Write the thesis paper.
(the thesis paper)

The timing is based on 20% of the development being done during normal working hours in order to meet with the experts. The remaining 80% was done on off-hours.

The thesis was delayed a total of 10 months trying to locate and qualify a suitable expert system shell. Since the initial evaluation in 1987, there are multiple expert system shells available that might fit the needs of this application. Once NEXPERT was qualified, the development proceeded according to the original timing.

CHAPTER 4

RESULTS

In preparation for developing the expert system using INSIGHT2+, a test scheme was developed to verify the functionality of INSIGHT2+. At that time 30% of the commands were not functioning according to the documentation. After evaluating two more versions of INSIGHT2+, the functionality required for the project was not present, so it was decided not to use this tool.

Without the availability of a viable expert system tool, a custom expert system shell was designed and development was started. When funding was available at the end of the year and a Macintosh SE computer was available for use, the NEXPERT expert system tool was evaluated and purchased.

Development of the knowledge base was done on the Macintosh and executed on the VAX, so a communications path was setup between the Macintosh and the VAX. Since the expert system tool was written in C, the process control application was written in Fortran, and the VAX environment uses Pascal, interface routines were written to allow all

data types to be passed between languages.

The interface routines were developed between the PDP process computer and the VAX expert system. It allows a command to be sent to the remote system, where data are collected, converted to a site independent format, and sent to the host, where it is passed into NEXPERT.

In order to allow for a robust knowledge base test scheme, a generic network simulation and a NEXPERT test environment were developed on the VAX. This environment can monitor and control the knowledge base in an interactive or batch command mode. Thus a command file can be created to test a specific rule and then to verify that the state of the knowledge base is correct (See Figure 4.1).

```
!
! HYPOTHESIS: Application disk headers are low
! KB:         kb_dir:system.kb
! Author:     Rich Winslow
! Date:       4/23/88
!
! DESCRIPTION: Application disk headers are set to only 5% free
!
     1 PDP
     1 DU0:  14.1  11.0
     1 DU1:  15.0   5.0   ! Application disk headers are low
@kb_test:act.test
@kb_test:rmd.test
@kb_test:copy_memory.test
@kb_test:dev.test
```

Figure 4.1

The expert system was developed from the interviews with the 3 other experts and the users. As the expert system executes, it asks the user for the process control site to investigate, the name of the production facility, and which terminal the user is logged into. The expert system was developed to investigate a broad spectrum of possible problems with the process control application. This includes hardware errors, software errors, and operator errors.

The knowledge base was modularized into separate knowledge base files for ease of understanding and maintenance. These files are loaded into the memory and executed as needed during the inferencing process. The expert system is driven by a combination of forward and backward chaining. For each conclusion that must be relayed to the user, a text file is displayed with 4 descriptions. The text describes the problem, an explanation of how the expert system made this conclusion, the possible causes of the problem, the solutions to the problem, and the implications of not solving the problem (See Figure 4.2).

Application disk file header count is low
PB: is assigned to DU1:

Explanation:

There are only 5 % free disk file headers.
The disk should not be less than 25 % free.

Possible causes:

- Disk was sized with too few headers.
- Not able to transfer batch history files to the VAX computer.
- New distribution was sent without purging the old distribution.
- Process history files grew too large.

Solutions:

- Scan the DU1: disk for old files.
- Purge the application files.
- Ask the software developer to cleanup the [group_uic,10] directory.
- Ask the system manager to resize the disk with more file headers. The system manager must backup the disk, initialize it with more headers and then restore the backup without re-initializing the disk.

Figure 4.2

A full test scheme was developed to test each rule in the knowledge base. This includes a test command file, a set of network simulation files, a set of test database files, and a set of test application shared memory images.

The expert system will be considered for release after additional reviews with the chemists and production operators. The expert system's current knowledge is based on an outdated design, which is over a year old. Since the design was documented in the thesis proposal, it was not updated as it would for a application not related to a thesis. Thus, additional knowledge must be included before it is

released.

CHAPTER 5

CONCLUSION

5.1 AUTOMATING AN EXPERT SYSTEM

When a majority of an expert system gathers its data directly from the application software and not from the user, the complexity of the expert system development increases along with the reliability of the data and the ease of use for the end-user. When sets of data are imported from the application software, there is a set of application routines to get the data, possibly a network interface command to request the data, a routine to prepare the data to be sent to the expert system, and a set of simulation files to simulate the data in the test scheme. All of this code would not be used by the expert system if the user were asked to input the data. However, the BMCS expert system imports data in more than 1000 property slots, which is not feasible for a user to have to enter.

Due to the extensive error checking in the application software and the expert system interface routines, the data that enter the expert system have a high and static certainty factor. The expert

system does not have to include rules to determine the reliability of the data or to deal with its uncertainty.

When the data come from the application data structures, it is not as visible to the experts. The knowledge engineer had to take the knowledge described by the experts and find equivalent and in some cases more reliable data from the application data structures. Thus there was an extra round of discussions to be sure that the data being used coincided with the expert's thoughts. This helped to heighten the experts' and the users' awareness and confidence of the application's capabilities.

The addition of the network interface to pass the data from the application to the expert system added another level of code that the expert system had to understand. Thus, a knowledge base was developed to diagnose network communication problems. The knowledge base is automatically loaded and started if an error status is returned from a network communication (e.g. connect to node, send, or receive).

5.2 USER DISPLAYS

A major portion of the knowledge base coding, as opposed to the knowledge acquisition and testing, was spent creating conclusion text files that are displayed to the user when the expert system has found something significant (See Figure 4.2). Each conclusion text file included 4 descriptions:

1) A one line definition of the problem.

2) An explanation of how the expert system concluded this.

3) A description of what the possible solutions are to the problem.

4) A description of the implications of not solving the problem.

This format followed the format of the troubleshooter's log book, which preceded this expert system. The brief problem definition is in bold as the first line in the conclusion text to get to the point so that the experienced user can note it and continue. The explanation serves as a documentation technique for the rules related to it, but also it allows the user to challenge the expert system. If the users feel the explanation doesn't adequately support the stated problem, they can make an intelligent decision to ignore it. Without the explanation, the user must blindly trust the knowledge base. This may lead to an erroneous action if the knowledge base is outdated, and it may prevent the knowledge base from being improved.

Each problem requires some action, whether it is something the user executes or when someone else must be called. The solution description serves this function. However, it may not be possible to provide the proper solution in a timely fashion. The implication description is used to let the user know the consequences of not solving the problem completely.

The knowledge acquisition process and the creation of good comprehensible text displays makes a knowledge engineer feel like a technical writer developing an automated documentation aid.

5.3 OPERATOR'S PERCEPTION OF THE PROBLEM

One valuable section of the troubleshooter's guide book was the description of the operator's perception of the problem along with the description of the actual problem that the troubleshooter found. This enabled troubleshooters to use the operator's accounting of the problem to reference the list of actual causes during future troubleshooting sessions.

The expert system was developed using the expert's description of the problem with the actual causes, but it does not include the relationship between the user's perception of the problem and the actual causes. The absence of the user's perception is acceptable if the user's knowledge is limited or tends to confuse the issue, but the expert system should have the capability to allow user input to help focus the expert system on the problem as perceived by the operator. The user interaction is useful after the automated interface has exhausted itself. The complexity of the application is such that a user typically does not comprehend underlying factors and relationships (e.g. computer hardware errors or the relationship between a solution flowrate and a sensor reading), and thus their perception of the problem may be a side-effect or even unrelated to the problem. Once the expert system knows the state of the underlying factors and relationships and still does not have a conclusion, the expert system can take input from the operator. When the expert system evolves to the point where operator input is used, these relationships must be described within the knowledge base. This will raise the expert system's worth to a new level. The expert system was designed in levels of intelligence:

1) simple binary problem solving (working or not working)

2) multiple conditions within a specific area of focus
   (the pump is off-aim and the flowrate check is marginal)

3) multiple conditions between specific areas of focus
   (the electrode is off-aim, electrode testing is OK,
   and the mixer speed is low)

4) multiple conditions between seemingly unrelated areas of focus
   (the kettle emptied too soon and a disk had excessive errors)

5) a mixture of operator observations and known conditions
   (the operator noticed that the equipment didn't turn on at
   the appropriate time and the final batch analysis was off aim)

For the purposes of the thesis, only the first three levels were fully incorporated into the expert system. The fourth level is included by describing multiple problems to the user, but not writing rules to explicitly deal with the interactions. The other levels will be included at a later date.

5.4 KNOWLEDGE BASE TESTING

As with conventional programming, testing and validation is a very important aspect of knowledge base development. In order to trust the expert system, one must provide a comprehensive test scheme, which will test a rule when it is created or modified, when related rules are added or modified, or when someone challenges the accuracy of a rule or set of rules. The test scheme must be able to accurately simulate the environment in order to control each piece of data that the knowledge base uses. With automated data acquisition, the source data files, memory, and network communications had to be simulated. The time required to incorporate these simulation techniques is justified in comparison to the cost of implementing a poorly tested

knowledge base.

The non-sequential nature of a rule base makes its understanding and maintenance more difficult. Rules are not evaluated sequentially from the top of the rule base to the bottom; they are evaluated according to their relationship with and the current state of other rules. The relationships and states of rules can change dynamically. Thus the need for a well designed and maintained test scheme and good documentation is a stronger issue for expert system development than for conventional programming.

User mistrust in this expert system could jeopardize its effectiveness and the implementation of future expert systems. The categorization of this software as an "expert" system naturally increases the perception that the software will be "perfect" and decreases the tolerance for error. The operators understand that the development of the expert system is an evolution, which is bound to need improvement, but the operators are more sensitive to errors that they perceive to be blatant. Thus, testing appears to be more critical for expert systems than for conventional programming.

5.5 KNOWLEDGE BASE DESIGN ISSUES

Throughout the development of the process control application and prior to the expert system development, a large percentage of the effort went into the development of equipment calibration and test routines to minimize the possibility of losing a production batch. This includes mathematical and statistical sampling and analysis,

event logging (e.g. communication errors and equipment calibrations), and extensive process history logging. As a result, the expert system was supplied with more concrete evidence than most, applications and the evidence was already condensed down to the last stage before interpretation. As a result, the object structures, the rules, and the modularization of the knowledge base was simplified. This simplification will make the knowledge base easier to understand and maintain. The flexibility and efficiency of the NEXPERT callable interface made it easy to organize the information gathering, so the algorithmic and statistical processing was done by conventional programming and the decisions were made by the inference engine.

5.5.1 The Knowledge Engineer

According to the literature [Wate 86], the knowledge engineer should not be the expert, because too much might be taken for granted and not be included in the knowledge base. This is also true with conventional programming if one programmer works on a project unassisted. The programmer might not include enough data verification and/or he/she might not fully test the routines. One solution in both cases is to work with a team of developers and users. The opinions of others can fill in the missing pieces and enhance the resulting product.

If a team of developers and users can complement the knowledge engineer, it can be beneficial for the expert to be the knowledge engineer. An important aspect of any project's success is based on the enthusiasm and "buy-in" of those involved. If the expert chooses

to be the knowledge engineer, there will be a high level of enthusiasm and desire to make the project work and to complete it, and the issues of expert resentment and fears of replacement by a machine are not factors.

Thus the use of one isolated knowledge engineer is more the issue than the use of the expert as a knowledge engineer. With use of multiple knowledge engineers and/or the proper communications between the engineer(s) and the users, the resulting expert system will not suffer from a lack of knowledge.

5.5.2 The Expert

The literature states that expert system development should avoid using multiple experts due to the problems of contradictions between experts and the amount of time required to interview experts [Harm 85]. This simplification will eliminate valuable information and adversely narrow the knowledge engineer's perspective. The expert system project can be organized to benefit from multiple experts without unresolvable contradiction and extensive development time. The knowledge engineer can select experts that can cooperate to form a consensus. If this is not possible, then the knowledge engineer must choose the strongest possibility and/or incorporate each of the opinions into the knowledge base. Multiple knowledge engineers may be necessary to complete the expert interviewing and/or some of the experts could become the knowledge engineers with a lead knowledge engineer to resolve conflicts, but the careful selection of multiple experts can provide a richer knowledge base than limiting the scope to

one expert.

5.5.3 Experts Who Understand The Expert System Tool

One of the design techniques under investigation was the difference between the use of experts who worked directly with the expert system tool and those who did not. Some literature states that experts with an understanding and access to the expert system tool will leave out knowledge that is not easy to represent in the tool [Wate 86]. As with the issue of the knowledge engineer serving as the expert, a team of cooperating engineers and users will minimize the use of these short cuts.

In fact, with the BMCS expert system, the experts with knowledge of the tool created deeper knowledge than those who did not know the tool. The expert who didn't know what the tool could do provided shallower information than those who understood the tool's representation and generalization techniques. It meant the knowledge engineer had to repeat discussions with the experts to get the representation correct. This coincides with conventional programming, where the user will ask for less from the project until the prototypes demonstrate the capabilities of the software and the developers. Users who have an understanding of computers and programming will give a more complete description of their needs.

5.6 FACTORS THAT BIASED THE CONCLUSIONS

These conclusions were based on a single project that may not be

considered typical. The project's team members (engineers, experts, and users) have worked well together for several years now; the plant site involved has put their employees through several training seminars to address organizational and communication skills and the use of concepts of project buy-in and reaching a consensus.

The fact that this project was linked directly with the thesis affected the way the project was executed. The project timing was condensed to provide milestones for the thesis. With the lead engineer as the author of the thesis, a larger portion of the project was done by the author on off-hours. Also, the development of the knowledge base was breadth first across multiple knowledge domains within the process control application instead of deeper knowledge over a smaller number of domains.

CHAPTER 6

SUMMARY

The added complexity to code and test the automated sections of the expert system were justified by the large amounts of data that the expert system was able to handle, the increased reliability of the data, and the ease of use for the end-user.

The general project development techniques used for conventional programming projects coincide with the development of expert systems. The use of good people (e.g. experts, engineers, and users), teamwork, and organization will minimize the problems with knowledge engineers as experts, with experts understanding the tools, and the use of multiple experts.

The selection of a lead knowledge engineer with the right personality and work skills is similar to the selection of a system analyst in a conventional programming project. The knowledge engineer must be able to communicate well with the expert(s) as well as the end-user(s), he/she must be able to act as a mediator to bring about consensus when contradictions arise, and he/she must be able to

organize and document the knowledge for easy understanding and maintenance. However, the lead knowledge engineer must have the above skills as well as the organization skills to keep the project on schedule and on focus. With the proper selection of knowledge engineers, the problems of weak and contradictory knowledge can be minimized.

The use of NEXPERT with development on the Macintosh was a good solution to the tradeoff between reduced hardware costs using existing equipment and having a good expert system tool. This will leave a favorable impression with management on the viability of expert systems that will allow for future purchases of hardware to improve the development environment.

APPENDIX A

GLOSSARY

Augmented rule control

Augmented rule control is the ability of an expert system shell to perform both forward and backward chaining operations on the same rule.

BMCS

This is the acronym for the Batch Manufacturing Control System application.

Batch chemical process

A facility that controls the mixing of chemicals into kettles. A kettle of material is known as a batch or a finite amount. In contrast there is the continuous chemical process that doesn't separate the material into batches but instead works with a continuous flow (in-line chemical injection).

Detached process

A detached process is a VAX/VMS term for a program that executes in the background. This is similar to a computer batch job without a log file. A detached process typically does not communicate directly with the user via a terminal, instead it communicates via mailbox messages and data files. Print symbionts and message routers are examples of detached processes.

Engineering Units

An engineering unit is the unit of measure that a user would relate to (e.g. pH, seconds, %). This is in contrast to raw data read from a sensor. Sensor readings are integer values that must be converted to engineering units before they can be used.

Ethernet

Ethernet is a network protocol developed by Xerox and implemented by many computer manufacturers including DEC. Ethernet describes the precise way data will be formed into packets and sent over a logical link between two devices. Ethernet is not a description of the physical link between two devices. Baseband and Broadband are examples of physical link descriptions.

Integrated expert system

An integrated expert system is an expert system that is capable of passing information to and from sources external to itself (e.g. a database or file, the operating system, or a network). The level of integration can vary from data transfers between files to the point where the expert system can be fully imbedded in an application's source code.

Knowledge engineer

The knowledge engineer is a person who designs and develops a knowledge base by learning as much as possible about the subject being captured in an expert system. This can include being assigned to the job previously or temporarily for this project, by interviewing experts and future end-users of the expert system, and by other methods of learning such as reading or taking courses. In loose terms this is an expert system's system analyst and/or programmer.

Mailbox communications

A VMS Mailbox is a message passing mechanism where two cooperating processes on the same computer can pass information in either direction. This is analogous to the UNIX pipe.

Object-oriented programming

Object-oriented programming is a programming environment, where the representation of physical objects and operations performed on them are grouped together. The grouping of all information about an object provides for a clearer understanding of the object's software representation and improves software maintenance.

PID

PID is a process control term for Proportional, Integral, and Derivative. It describes the type of control algorithm which is used by a programmable controller to maintain a process at a given aim point. Each letter represents a constant in the algorithm, which are varied to change the type of control which is needed.

Process (or task)

The word PROCESS has at least two meanings in this document. First there is the process (a computer program), which is run on a computer and second there is the chemical process, which is controlled by the computer. The chemical process is the sequence of operations that are executed to mix multiple chemicals together.

Process control environment

The process control environment is a term that is used in the process control application software. It describes all of the sources of data that is available to the software. This includes shared memory, database files, equipment I/O.

Process control system

The process control system is the PDP 11 computer and the programmable controller, which is interfaced to the PDP computer. Together the two hardware components control the chemical batch process.

Shell

An expert system shell is a software package which handles the control portion of an expert system. The knowledge engineer must develop the knowledge base to be run by the expert system shell. The shell can include terminal I/O, screen formats, explanation facilities, the inference engine, reporting, and testing facilities.

Subprocess

A subprocess is a computer process (or job stream) that is created (forked) by a parent process. The subprocess can only exist as long as the parent process is exists, but the parent can hibernate (suspend execution) until the subprocess terminates. Thus this is a temporary process.

APPENDIX B

ACKNOWLEDGEMENT

The expert system project was made possible by the contributions of Christopher Haupt, Mike Meynadasy, Beverly Fair, William McMaster, Joseph McDonough, James Watt, and Donald Ellsworth. With the help of those mentioned above and my thesis advisors John A. Biles and Patrick Meehan, the completion of this thesis has become a reality.

APPENDIX C

BIBLIOGRAPHY

Books

[Char 85] Charniak, Eugene, and McDermott Drew. Introduction to Artificial Intelligence. Reading, MA: Addison-Wesley Publishing Co., 1985.

[Harm 85] Harmon, Paul, and King, David. Artificial Intelligence in Business - Expert Systems. New York: John Wiley and Sons, Inc., 1985.

[Wate 86] Waterman, Donald A. A Guide to Expert Systems. Reading, MA: Addison-Wesley Publishing Co., 1986.

[Cloc 84] Clocksin, William F, and Mellish Christopher S. Programming in Prolog. New York: Springer-Verlag, 1984

[Barr 85] Barrett, Rosalind, Ramsay, Allan, Sloman, Aaaron. POP-11 A Practical Language for Artificial Intelligence. London: Ellis Horwood LTD, 1985.

[Hols 85] Holsapple, Clyde W, and Whinston, Andrew B. Manager's Guide to Expert Systems Using GURU. Homewood, IL: Dow Jones-Irwin, 1985.

[Fors 84] Forsyth, Richard, Expert Systems Principles and case studies. New York, NY: Chapman and Hall Computing, 1984.

[Klin 85] Kline, Paul J, and Dolins, Steven B. Choosing Architectures for Expert Systems. Griffiss Air Force Base, NY: Rome Air Development Center, 1985.

Software Product Documentation

[1st 86] 1st Class Reference Manual. Wayland, MA: Programs in Motion, Inc., 1986.

[Insi 87] Insight2+ Reference Manual. Indialantic, Florida: Level Five Research, Inc., 1987.

[Nexp 88] NEXPERT Reference Manual. Palo Alto, CA: Neuron Data, 1988

[Lisp 86] VAX LISP Reference Manual. Maynard, MA: Digital Equipment Corporation, 1986.

Periodicals

[Chow 86] Chowdhury, Jayadev. Troubleshooting comes online in the CPI. PP 14 Chemical Engineering, October 13, 1986.

[Beck 87] Beckert, Beverly A. Expert Systems: Ready for Industry. PP 44 CAE, January, 1987.

[Solo 87] Soloway, Elliot, and Bachant, Judy, and Jensen, Keith. Accessing the Maintainability of XCON-in-RIME. PP 824 Expert Systems, Fall, 1987.

Presentations

[Rowi 86] Rowin, Duncan. Review of the Falcon Expert System. Fall 1986.

INDEX